

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Cloud Storage**

propusă de

**Eduard Marcu**

**Sesiunea:** iunie/iulie, 2023

Coordonator științific

**Lect. Dr. Vârlan Simona-Elena**

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**

# **Cloud Storage**

**Eduard Marcu**


**Sesiunea: iunie/iulie, 2023**

Coordonator științific

**Lect. Dr. Vârlan Simona-Elena**

Avizat,  
Îndrumător lucrare de licență,  
Lect. Dr. Vârlan Simona-Elena.

Data: .....22.06.2023.....

Semnătura: .....


### **Declarație privind originalitatea conținutului lucrării de licență**

Subsemnatul **Marcu Eduard** domiciliat în **România, jud. Iași, mun. Iași, strada Clopotari, nr. 48, bl. 674, et. 1, ap. 6**, născut la data de **03 martie 2001**, identificat prin CNP **5010303226722**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2023, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Cloud Storage** elaborată sub îndrumarea doamnei **Lect. Dr. Vârlan Simona-Elena**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: .....22.06.2023.....

Semnătura: .....

### **Declarație de consimțământ**

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Cloud Storage**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Eduard Marcu**

Data: .....22.06.2023.....

Semnătura: *Marcu*.....

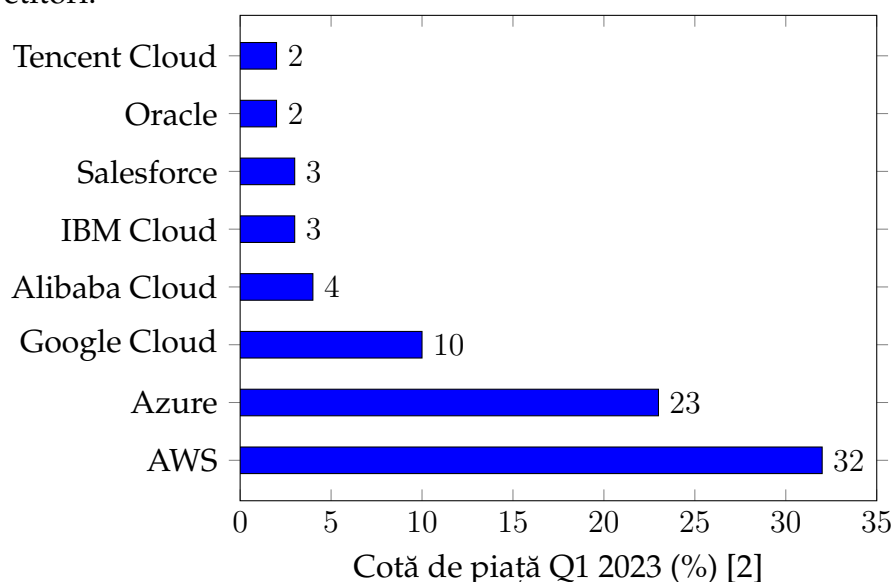
# Cuprins

<b>Motivație</b>	<b>2</b>
<b>Introducere</b>	<b>4</b>
<b>Contribuții</b>	<b>5</b>
<b>1 Aplicații similare</b>	<b>6</b>
1.1 Google Drive . . . . .	6
1.2 Mega . . . . .	7
<b>2 Tehnologii utilizate</b>	<b>8</b>
2.1 TypeScript . . . . .	8
2.2 React . . . . .	8
2.3 Chakra UI . . . . .	9
2.4 Axios . . . . .	9
2.5 React Query . . . . .	9
2.6 Express.js . . . . .	10
2.7 Firebase . . . . .	10
2.8 Prisma . . . . .	10
2.9 Google Cloud Storage . . . . .	10
<b>3 Proiectare și implementare</b>	<b>11</b>
3.1 Arhitectura . . . . .	11
3.2 Implementarea . . . . .	13
3.2.1 Gestionarea sesiunii și securității . . . . .	13
3.2.2 Reprezentarea sistemului de fișiere . . . . .	14
3.2.3 Încărcarea . . . . .	15
3.2.4 Proprietățile fișierelor și funcționalitățile . . . . .	19

3.2.5	Căutarea . . . . .	23
3.2.6	Curățarea . . . . .	24
<b>Concluzii</b>		<b>25</b>
3.2.7	Direcții viitoare și limitele aplicației . . . . .	25
3.2.8	Opinie personală . . . . .	26
<b>Bibliografie</b>		<b>27</b>

# Motivație

Dimensiunea pieței globale de stocare în cloud a fost evaluată la 90,17 miliarde USD în 2022 și se estimează că va crește până la 472,47 miliarde USD până în 2030 [1]. Această industrie este în continuă creștere și dezvoltare, având un număr mare de competitori.



Trăind într-o eră digitalizată unde suntem în permanență înconjuțați de tehnologie, nevoia serviciilor de stocare în cloud devine din ce în ce mai mare. Transferul de fișiere și documente, redactarea documentelor în mediul online, găzduirea de aplicații web, serviciile de streaming de muzică și filme, stocarea pe termen lung sau scurt a datelor cu caracter personal sau a diverselor informații ale unei companii, toate aceste funcționalități au la baza stocarea în cloud. Avantajele care vin cu folosirea acestei tehnologii sunt de asemenea bine primite de către utilizatori, aceștia nefiind nevoiți să creeze un buget pentru dispozitive de stocare sau să investească timp în gestionarea corectă a resurselor. Aceste aspecte scot în evidență importanța serviciilor de stocare în cloud cât și utilitatea și diversitatea acestora.

Google Drive, Dropbox, Microsoft OneDrive, Apple iCloud Drive sunt aplicațiile cele mai populare în contextul serviciilor de stocare, partajare și descărcare a fișierelor,

având un număr impresionant de utilizatori (peste două miliarde de utilizatori) [3].

Din dorința de a vedea în detaliu cum sunt protejate și prelucrate fișierele, cum se previne conceptul de „race condition” în lucrul cu acestea, cum se realizează partajarea, descărcarea și încărcarea acestora, am luat decizia de a crea o aplicație web de gestionare a fișierelor cu scopul de a intra în amănuntele tehnicilor folosite pentru stocarea în cloud, având de asemenea ca termen de comparație acești giganti ai industriei.



# Introducere

În era informațională de astăzi, cererea pentru soluții eficiente de stocare a datelor a crescut într-un mod fulgerător. Odată cu progresul rapid al tehnologiei, stocarea în cloud a apărut ca un instrument esențial în societatea de astăzi, satisfacând dorința din ce în ce mai mare de acces la resurse de la distanță.

Cloud Storage este o aplicație web, orientată către simplitate, ce se prezintă ca o alternativă a aplicațiilor web mai populare precum Google Drive, oferind utilizatorilor un spațiu de stocare sigur în cloud într-un mod accesibil și convenabil, acesta având posibilitatea de a încarca, descărca și partaja resursele sale. Astfel, utilizatorii se pot elibera de constrângerile dispozitivelor de stocare locale și a metodelor tradiționale de partajare a fișierelor.

Această teză explorează dezvoltarea unei aplicații web de stocare în cloud care valorifică puterea tehnologiilor precum: React, Chakra UI, Axios, React Query, Express.js, Firebase, Google Cloud Storage și Prisma, ce au un rol important în a oferi utilizatorului o experiență cât mai plăcută și intuitivă din punct de vedere vizual cât și interactiv.

În primul capitol, teza prezintă câteva aplicații existente, axându-se pe funcționalitatea de partajare a unei resurse, deoarece aici se văd cele mai mari diferențe în implementări.

Capitolul intitulat „Tehnologii utilizate”, face o scurtă prezentare asupra acestora și oferă motivația ce a stat în spatele alegerilor făcute.

În al treilea capitol, se descrie arhitectura aplicației și se explică temeinic logica ce stă în spatele funcționalităților.

# Contribuții

În cadrul dezvoltării acestei aplicații web, Cloud Storage are patru componente ce stau la baza funcționării:

- Încărcare de fișiere și directoare.
- Descărcare de fișiere și directoare.
- Prevenirea conceptului de „race condition”.
- Reprezentarea fișierelor într-o manieră similară unui sistem de fișiere.

În cadrul acestei teze, cele patru componente vor fi prezentate într-o manieră detaliată, trecând prin procesul de gândire folosit pentru a ajunge la implementarea curentă a acestora. Prin aprofundarea acestui proces, teza își propune să ofere o relatare cuprinzătoare a modului în care componentele au fost concepute.

# Capitolul 1

## Aplicații similare

Aplicațiile de gestionare a fișierelor în cloud există pe piață de aproape două decenii, Google fiind printre primii pionieri ai acestei industrii cu aplicația Google Docs lansată în 2006 [4]. Astfel, Google a avut un rol semnificativ în popularizarea conceptului de prelucrare a fișierelor în cloud, iar prin tranziția de la Google Docs la Google Drive, această companie și-a consolidat poziția în această industrie.

Prin urmare, astfel de aplicații au avut timp să se maturizeze și ca o consecință marea majoritate a aplicațiilor de acest fel au în principiu aceleași funcționalități cu mici diferențe. În cele ce urmează teza se va axa pe sistemul de partajare al resurselor deoarece aici găsim cele mai mari diferențe.

### 1.1 Google Drive

Procedeul prin care poți partaja o resursă în Google Drive este amplu, oferind utilizatorilor o gamă largă de opțiuni. Partajarea are loc prin crearea unui link ce poate avea mai multe proprietăți. Acest link poate fi nerestricționat, prin urmare oricine are link-ul poate accesa resursa, sau restricționat prin adăugarea de adrese de email la o listă de acces. Google adaugă un pas în plus accesului restricționat, având posibilitatea de a asigna și nivelul de acces: cititor, comentator, editor.

Fiind un jucător important cu vechime, Google Drive satisface majoritatea nevoilor unui posibil utilizator, singurul dezavantaj putând fi că după ani de dezvoltare, interfața a început să devină încărcată cu redundanțe.

## 1.2 Mega

Similar cu Google Drive, Mega are de asemenea și el două metode de a partaja o resursă prin link: restricționat sau nerestricționat, însă accesul restricționat lasă de dorit. Ideea principală este de creare a unei chei pentru a accesa resursa. Cu alte cuvinte dacă dorești să trimiți un fișier, trebuie să trimiți și cheia. Prin urmare, receptorul poate la rândul lui să trimită mai departe informația fără a întâmpina dificultăți.

# Capitolul 2

## Tehnologii utilizate

Cloud Storage este o aplicație web ce se folosește de numeroase tehnologii pentru a livra o experiență adecvată utilizatorilor. În cele ce urmează voi justifica utilitatea acestora.

### 2.1 TypeScript

TypeScript este un limbaj de programare open source ce are la bază JavaScript. Prin urmare, beneficiază de tot ce oferă JavaScript și adaugă funcționalități noi cum ar fi tipurile de variabilă ce îmbunătățesc experiența de dezvoltare, deoarece prevenirea și prinderea de erori pe bază de tip devine mai ușor de realizat în timpul dezvoltării proiectului.

De asemenea, acest limbaj de programare vine cu proprietăți precum interfețe, ce oferă unui programator posibilitatea de a reprezenta cu ușurință tipuri complexe [5].

### 2.2 React

React [6] este o bibliotecă open source, realizată de Facebook, pentru JavaScript și este de asemenea compatibil cu TypeScript. Popularitatea sa este redată de diversele unelte ce fac construirea unei interfețe mai comodă [7]:

- Virtual DOM (Document Object Model): React se folosește de un DOM virtual care menține interfața utilizatorului în memorie. Astfel, atunci când există modificări în starea aplicației, React actualizează DOM-ul virtual și calculează diferențele dintre acesta și DOM-ul real al navigatorului web. În acest mod se obține mulțimea

minimală de schimbări necesare actualizării DOM-ului real al navigatorului web. Acest proces rezultă într-o creștere semnificativă în performanță, deoarece numărul de manipulări reale ale DOM-ului scade [7] [8]. Într-o aplicație web ce presupune parcurgere prin directoare, această funcționalitate îmbunătățește semnificativ experiența utilizatorului.

- Hooks: oferă o modalitate de a atribui componentelor funcționale propria stare, simplificând logica acestora și eliminând necesitatea de a scrie componente de tip clasă [7]. În cadrul proiectului această unealtă a fost folosită pentru o mai bună comunicare între componente și pentru gestionarea stării proiectului.
- JSX: o extensie JavaScript ce permite scrierea unui limbaj asemanător HTML-ului înăuntrul codului JavaScript [7], facilitând astfel un cod mai ușor de citit și menținut.

## 2.3 Chakra UI

În realizarea unei interfețe grafice, ușor de utilizat, există multe decizii ce trebuie luate în legătură cu aspectul acesteia, reprezentând o dificultate semnificativă în realizarea unei aplicații web.

Chakra UI este o bibliotecă de componente UI flexibile și customizabile ce ajută în ușurarea acestei dificultăți, scăzând drastic timpul necesar dezvoltării unui proiect[9].

## 2.4 Axios

Axios este un client HTTP bazat pe promisiuni pentru Node.js ce oferă un mod simplu de comunicare cu serverul [10]. Motivul principal pentru care am ales să folosesc axios, în cadrul aplicației Cloud Storage, a fost faptul că acesta se ocupă singur de convertirea de JSON, fapt ce face lucrul cu astfel de date convenabil.

## 2.5 React Query

React Query este o bibliotecă din React ce simplifică procesul de primire și trimitere a datelor cât și gestionarea stării aplicației [11]. Cea mai utilă și folosită unealtă

din cadrul acestei biblioteci a fost invalidarea de interogări, ușurând procesul de actualizare a datelor când acestea devin învechite în urma unor acțiuni ale utilizatorului.

## 2.6 Express.js

Express.js este un framework de Node.js pentru dezvoltarea unui server HTTP pentru o aplicație web [12]. Fiind de asemenea compatibil cu TypeScript, alegerea de a realiza un server în același limbaj cu limbajul folosit pentru client a fost una naturală.

## 2.7 Firebase

Firebase oferă diverse unelte puternice pentru autentificarea, gestionarea și autorizarea utilizatorilor pentru cereri REST [13]. Ocupându-se singur de aceste concepte, Firebase preia din diferitele dificultăți pe care un programator ar fi nevoit să le rezolve.

## 2.8 Prisma

Baza de date a aplicației web Cloud Storage este realizată prin Prisma, aceasta simplificând crearea, accesul și gestionarea unei baze de date [14]. În cadrul acestui proiect, Prisma a fost folosită pentru crearea unor tabele de gestionare a stării fișierelor și a unor link-uri către acestea.

## 2.9 Google Cloud Storage

AWS, Microsoft Azure și Google Cloud Storage [15] sunt cele mai mari servicii de cloud și indiferent de alegerea făcută, fiecare poate să stea la baza unei aplicații de gestionare a fișierelor [16].

Motivul principal pentru care aplicația web, Cloud Storage, are la bază serviciile oferite de Google, constă în oferta generoasă a acestora, având posibilitatea de a folosi gratuit, pe un termen de 90 de zile, platforma Google Cloud Storage.

# Capitolul 3

## Proiectare și implementare

### 3.1 Arhitectura

Aplicația web, Cloud Storage, de gestionare a fișierelor în cloud, este realizată pe o arhitectură client-server, unde clientul este construit prin React TypeScript, iar server-ul prin Express.js.

Nivelul întâi, din cadrul modelului C4, este alcătuit din utilizatori și aplicația în sine.

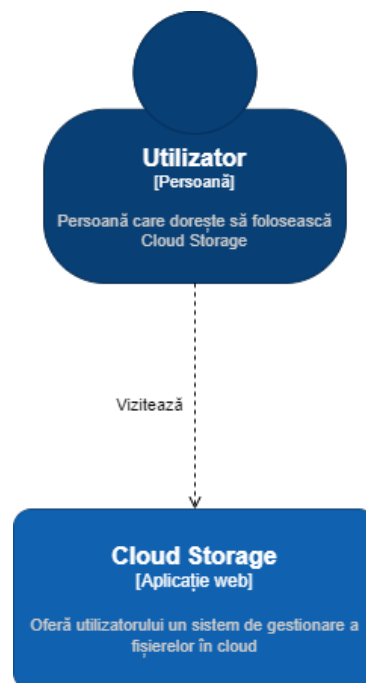


Figura 3.1: Nivelul 1 din diagrama C4

În nivelul al doilea, din cadrul modelului C4, putem vedea cum funcționează aplicația web într-un mod mai detaliat. Întâi, utilizatorul este nevoit să se autentifice,



după care este trimis pe pagina principală a proiectului. Din această pagină, au loc toate cererile API pe care le face clientul. Aceste cereri se duc către server, unde sunt validate, iar clientul primește un răspuns conform cererii. Server-ul comunică cu două recipiente de stocare a datelor:

- baza de date, realizată în Prisma, ce are scopul de gestionare a unor link-uri către resurse și a unor stări.
- bucket-ul din Google Cloud Storage, unde se află fișierele utilizatorului.

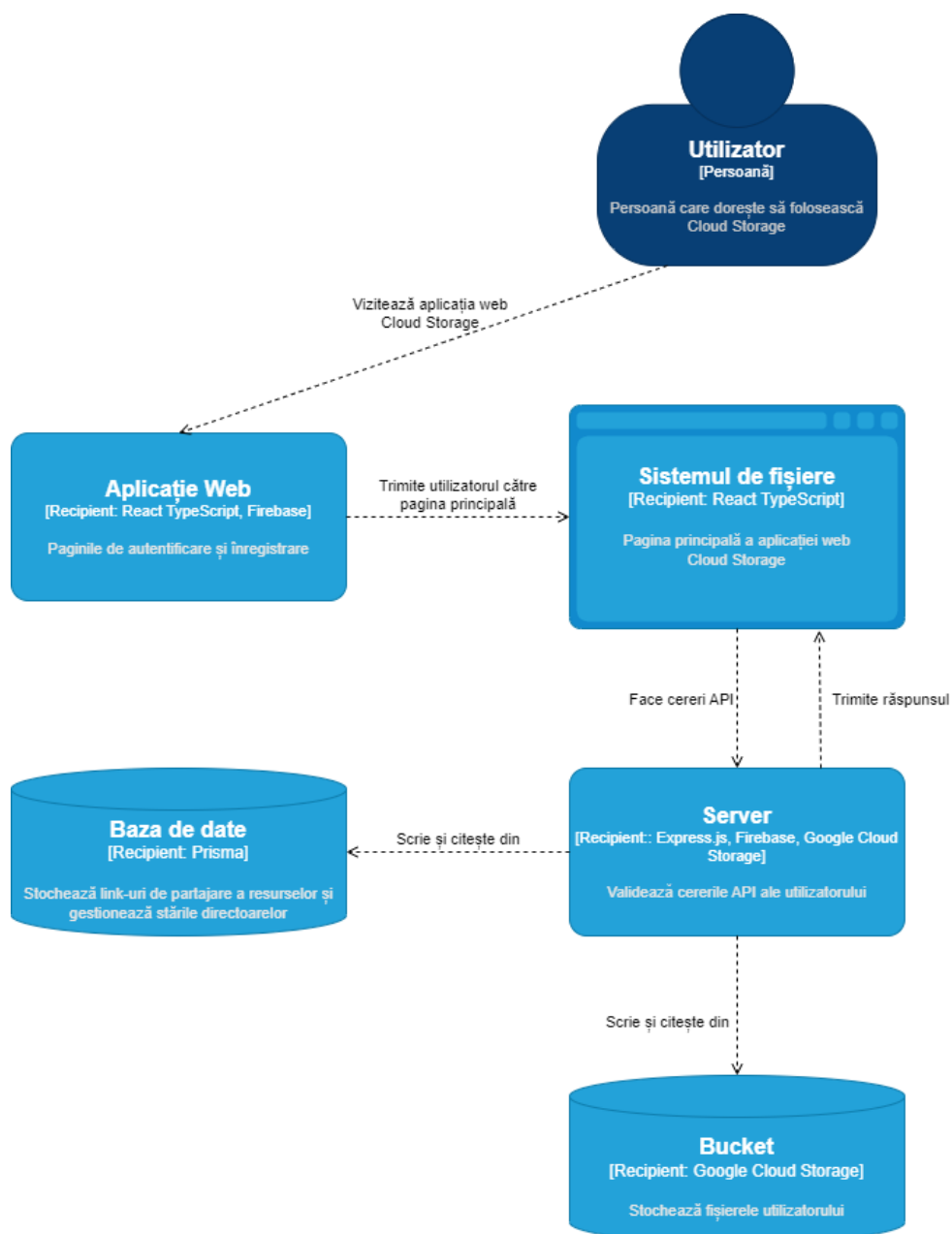


Figura 3.2: Nivelul 2 din diagrama C4

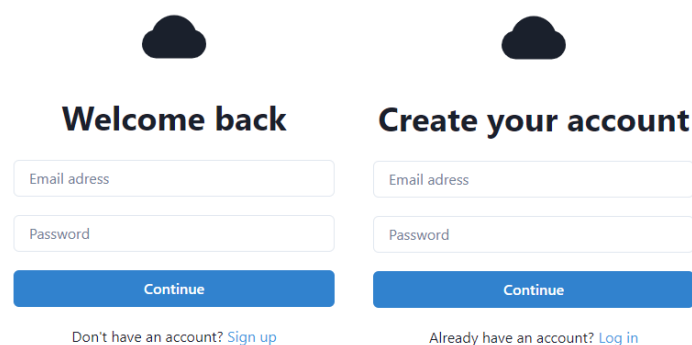
## 3.2 Implementarea

### 3.2.1 Gestionarea sesiunii și securității

Într-o aplicație de gestionare a fișierelor, un utilizator are un interes întemeiat în legătură cu securitatea datelor, acesta dorind să se asigure că datele sale se află într-un mediu securizat.

#### Autentificarea

Prima pagină pe care o vede un utilizator este cea care oferă două opțiuni: autentificare și înregistrare. Prin apăsarea oricărei opțiuni, utilizatorul este dus pe o nouă pagină unde i se prezintă formularul corespunzător opțiunii.





	
<b>Welcome back</b>	<b>Create your account</b>
<input type="text" value="Email adress"/>	<input type="text" value="Email adress"/>
<input type="text" value="Password"/>	<input type="text" value="Password"/>
<input type="button" value="Continue"/>	<input type="button" value="Continue"/>
Don't have an account? <a href="#">Sign up</a>	Already have an account? <a href="#">Log in</a>

Figura 3.3: Formulare de autentificare și înregistrare

Autentificarea și înregistrarea sunt realizate prin intermediul serviciilor Firebase. Cu alte cuvinte, Firebase se ocupă de toată logica ce poate veni cu aceste formulare, utilizatorul primind un mesaj corespunzător de succes sau eroare după completarea acestuia.

#### Sesiunea

Odată cu completarea cu succes a oricărui formular, utilizatorul este dus pe pagina principală a proiectului unde are loc stabilirea sesiunii. În primă fază va apărea o animație de așteptare ce oferă timp pentru stabilirea datelor necesare pentru crearea sesiunii.

După ce datele sunt stabilite, se face o verificare asupra acestora. Dacă utilizatorul a încercat să acceseze pagina principală într-un mod forțat, prin URL fără autentificare, acesta este dus înapoi către pagina de autentificare. De asemenea, o sesiune durează o

oră, deci în momentul în care a trecut o oră de la ultima autentificare, utilizatorul este dus înapoi pe pagina de autentificare.

## Autorizarea

Orice cerere de date sau prelucrare de date necesită două lucruri: sesiunea utilizatorului să fie validă și resursa asupra căreia se face cererea sau prelucrarea să fie într-adevăr a utilizatorului. Modul prin care este stabilit dacă o resursă aparține utilizatorului va fi discutată în subcapitolul următor.

### 3.2.2 Reprezentarea sistemului de fișiere

În Google Cloud Storage fișierele sunt reprezentate ca obiecte și nu există o ierarhie de directoare precum în Windows File Explorer. Cu alte cuvinte, un director este doar un fișier gol cu caracterul „/” pus la finalul numelui. Prin urmare, reprezentarea fișierelor sub o formă de sistem de fișiere necesită anumite prelucrări.

În aplicația web, Cloud Storage, un fișier are următoarea formă în Google Cloud Storage: „userIDuser@email.com/folderA/file.pdf”, unde „userIDuser@email.com/” va fi considerat directorul rădăcină pentru fiecare utilizator. Astfel, când un utilizator încearcă să ceară sau să prelucreze un fișier, se verifică dacă prin concatenarea identicatorului și email-ului, obținute din token-ul utilizatorului, obținem directorul rădăcină al resursei.

Obținerea fișierelor în forma lor ierarhică se realizează prin apelul către „app.get('/:currentPath', async (req: CustomRequest, res: Response)” din index.ts, unde „currentPath” reprezintă calea către un director. Acest apel preia toate fișierele din Google Cloud Storage care au ca prefix calea dată ca parametru al apelului.

Dacă calea reprezintă un director gol, acest apel returnează o listă goală. Dacă calea nu reprezintă un director gol, se adaugă într-o listă toate fișierele care au calea chiar parametrul apelului, adică cele care se regăsesc direct în directorul dat ca parametru. Pentru fișierele care au acest prefix, dar nu se regăsesc direct în directorul dat ca parametru, se adaugă în listă următorul director din calea fișierelor.

Evident, prin adăugarea de directoare în listă putem avea același director de mai multe ori. Prin urmare, se va realiza o filtrare pentru a rămâne doar cu directoarele unice.

Astfel obținem o listă unde fiecare element are următoarele proprietăți: id (întreaga

cale a fișierului), numele fișierului (fără cale), o variabilă boolean care este adevărată dacă este fișier, fals în caz contrar, și tipul fișierului.

În final se realizează o sortare asupra listei pe baza numelui elementului, dar condiționată de variabila boolean, astfel încât directoarele să apară înaintea fișierelor.

### 3.2.3 Încărcarea

În cadrul aplicației Cloud Storage, utilizatorul poate face următoarele operații de încărcare: încărcare fișiere, creare de director, încărcare de director.

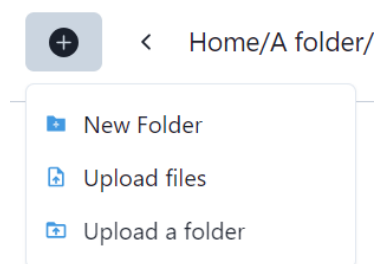


Figura 3.4: Meniul de încărcare

În următoarele subcapitole se vor găsi explicații pentru fiecare opțiune.

#### Încărcare fișiere

Dacă utilizatorul apasă pe opțiunea de încărcare de fișiere, o fereastră albastră va apărea pe ecranul acestuia. Această fereastră indică modul prin care utilizatorul poate face încărcarea de fișiere prin instrucțiunile găsite înăuntrul ferestrei.

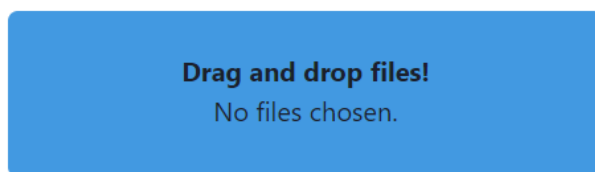


Figura 3.5: Spațiul de încărcare de fișiere

Această fereastră se folosește de HTML Drag and Drop API [17] și de DataTransfer: items property [18], pentru a putea prelua datele de intrare de la utilizator, adică fișierele. După ce am preluat datele de intrare, le parcurgem și cu ajutorul webkitGetAsEntry() [19] și getAsFile() [20] obținem un obiect de tip fișier pe care o să îl trimitem către server.

```

for (let i = 0; i < event.dataTransfer.items.length; i++) {
  (async () => {
    const item = event.dataTransfer.items[i];
    if (item.kind === 'file') {
      const entry = item.webkitGetAsEntry();
      if (entry?.isFile) {
        const someFile = item.getAsFile();

        if (someFile) {
          if (someFile.size <= maxSize) {
            newPostMutation.mutate({ myFile: someFile, path: currentPath });
          } else {
            ok = 0;
          }
        }
      }
    }
  })().catch((error) => console.log(error));
}

```

Figura 3.6: Parcurgerea datelor și trimiterea acestora către server

Odată ajuns în server, un fișier va fi încărcat în memorie și se va face o verificare pe baza numelui. Chiar dacă tot ce este încărcat în cloud va avea un cod numeric atribuit numelui, pentru a asigura unicitatea, ar fi deranjant pentru utilizator dacă în interfața acestuia se pot afla două fișiere cu același nume în același director. Prin urmare, pe baza extensiei fișierului, se face o verificare pentru a asigura și unicitatea numelui.

De asemenea, când un fișier se află în procesul de încărcare, se actualizează o tabelă numită `CurrentStateOnUpload` care are următoarele atribute:

- `id`: string = generat de Prisma.
- `userID`: string = identificatorul utilizatorului
- `fileName`: boolean = numele directorului în care este încărcat fișierul.

Datorită acestei tabele, operația de ștergere a unui director, ce are în ierarhia lui directorul unde se face încărcarea va fi imposibilă. După încărcarea fișierului în cloud, ștergem rândurile pe care le-am adăugat anterior.

O abordare anteriorară a acestei funcționalități a fost utilizarea unei etichete input de tip fișier ce avea atributul multiple. Astfel, când utilizatorul ar fi apăsât pe această etichetă, o fereastră cu fișierele lui din sistemul de operare ar fi apărut. Din această fereastră, utilizatorul putea să selecteze fișierele dorite pentru încărcare. Am considerat că acțiunea de a trage fișierele și a le plasa în spațiul de încărcare ar fi mai intuitivă pentru un utilizator decât procesul descris.

## Creare director

Când utilizatorul dorește să creeze un director prin intermediul interfeței aplicației web, o fereastră simplă va apărea pe ecranul acestuia, unde va fi nevoit să introducă numele directorului.



Figura 3.7: Crearea unui director

Mecanismul de verificare a unicității numelui directorului funcționează similar cu cel pentru încărcarea de fișiere, diferența constând în faptul că nu se va lua o extensie în calcul, ci doar numele. De asemenea, se folosește același proces pentru a ne asigura că nu poate exista o ștergere, dacă crearea directorului nu a luat sfârșit.

## Încărcare director

Dacă utilizatorul apasă pe opțiunea de încărcare director, o fereastră cu directoarele din sistemul lui de operare se va deschide. Această acțiune are loc prin utilizarea `window.showDirectoryPicker()` [21], ce va returna un obiect `FileSystemDirectoryHandle`, adică directorul ales.

Odată ce avem acest obiect, se apelează o funcție recursivă pentru a parcurge directorul în întregime. În această funcție recursivă, se parcurg valorile obiectului director de două ori. În prima parcurgere preluăm fișierele de pe nivelul curent și conținutul acestora pentru a le trimite către server.

Pentru a putea prelua conținutul unui fișier este nevoie mai întâi să avem un obiect fișier. Acest lucru se realizează prin `directory.getFileHandle(entry.name)` [22], unde `directory` este obiectul director de pe nivelul curent, iar `entry.name` este numele fișierului.

După ce am obținut un obiect fișier ne folosim de `getFile()` [23], pentru a prelua conținutul acestuia, și trimitem fișierul către server.

În a doua parcurgere, preluăm directoarele de pe nivelul curent pentru a le trimite către server și apelăm din nou recursia pentru fiecare director, folosindu-ne de `directory.getDirectoryHandle(entry.name)` [24]. Motivul pentru care valorile directorului de

pe un nivel se parcurg de două ori este pentru a asigura corectitudinea ierarhiei acestora. Orice fișier sau director încărcat va avea un cod numeric atribuit numelui, prin urmare dacă nu ar exista două parcurgeri ar fi imposibil de ținut minte calea fișierelor în recursie.

```
for await (const entry of directory.values()) {
  if (entry.kind === 'file') {
    const fileHandle = await directory.getFileHandle(entry.name);
    const fileContents = await fileHandle.getFile();
    if (fileContents.size < 10 * 1024 * 1024) {
      await axios.post(
        `/uploadFolder/${path + Date.now() + '-' + entry.name}`,
        fileContents,
        axiosConfig
      );
    }
  }
}
for await (const entry of directory.values()) {
  if (entry.kind === 'directory') {
    const tmpCode = Date.now();
    axios.post(`/uploadFolder/${path + tmpCode + '-' + entry.name}`, null, axiosConfig);
    await listEntries(
      await directory.getDirectoryHandle(entry.name),
      path + tmpCode + '-' + entry.name + '*',
      token
    );
  }
}
```

Figura 3.8: Parcurgerea datelor din funcția recursivă

În server, toate fișierele și directoarele din directorul primit de la utilizator, vor fi încărcate într-o zonă temporară, inaccesibilă utilizatorului. După ce întregul director a fost încărcat acesta va fi mutat în locația inițială. Bineînțeles, înainte să fie mutat, se realizează o verificare pe baza numelui pentru directorul dat ca parametru, pentru a obține un nume unic. Această verificare are loc doar pentru directorul părinte, deoarece celelalte fișiere vor avea deja un nume unic.

Motivul din spatele acestei implementări este pentru a nu lăsa utilizatorul să facă diverse operații pe o resursă care se află în procesul de încărcare. De menționat, mutarea unui fișier nu este o operație costisitoare, nefiind influențată de dimensiunea acestuia.

Dacă utilizatorul șterge directorul în care se face încărcarea directorului dat, pot exista două rezultate. Fie directorul nu a ajuns să fie complet încărcat în cloud, prin urmare ștergerea va avea loc și se va arunca o eroare în server când încercăm să mutăm directorul, fie directorul este încărcat în cloud și urmează să fie mutat, prin urmare ștergerea nu poate avea loc. Gestionarea acestei operații se realizează cu tabela `CurrentStateOnUpload`.

O abordare anterioară a acestei funcționalități de încărcare a fost prin utilizarea proprietății `webkitdirectory` [25], dar aceasta venea cu un dezavantaj semnificativ. `Webkitdirectory` oferă informații strict asupra fișierelor dintr-un director. Informațiile includeau calea fișierului în director, deci se putea realiza structura directorului în server, dar dacă în directorul dat de utilizator ar fi existat directoare goale, această proprietate nu ar fi oferit nicio informație asupra lor. Cu alte cuvinte, respectarea ierarhiei directorului nu ar fi fost posibilă.

În aplicația web, Cloud Storage, utilizatorul poate crea un director prin intermediul interfeței, dar această funcționalitate ar trebui să fie ceva bonus pentru utilizator, nu rezolvarea unui neajuns.

### 3.2.4 Proprietățile fișierelor și funcționalitățile

#### Reprezentarea datelor

Amintim din capitolele anterioare că un fișier sau un director are următoarele proprietăți:

- `id`: string = calea.
- `name`: string = numele.
- `isFile`: boolean = adevărat dacă este fișier, fals în caz contrar.
- `fileType`: string = tipul.



Figura 3.9: Fișiere și directoare

Pe baza proprietăților `isFile` și `fileType`, se generează aceste componente astfel încât să aibă o reprezentare grafică sugestivă. În cazul în care tipul de fișier este unul necunoscut acesta va primi reprezentarea grafică pe care o are documentul „this.docx” 3.9.



Dacă utilizatorul apasă pe o componentă director, se va apela „app.get('/:currentPath', async (req: CustomRequest, res: Response)”, obținând direcțiile și fișierele din acel director.

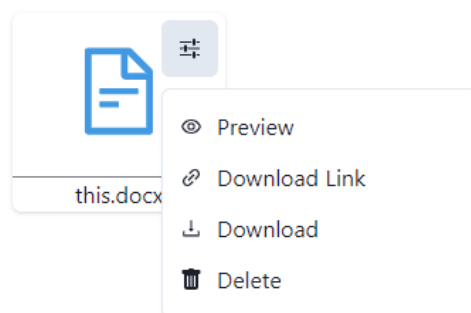


Figura 3.10: Opțiunile unui fișier

Dacă utilizatorul apasă pe butonul din dreapta sus, de pe o componentă, următoarele opțiuni vor apărea: previzualizare (doar dacă este fișier), descărcare, obținerea unui link pentru partajarea resursei, ștergere.

### Previzualizarea

Când utilizatorul selectează opțiunea de previzualizare, de pe un fișier, se apelează „app.get('/downloadFile/:downloadPath', async (req: CustomRequest, res: Response)” ce returnează un url către acea resursă [26]. Pe urmă, acest url este deschis prin intermediul Google Docs într-o fereastră nouă.

```
if (response.data) {  
  const viewerUrl = `https://docs.google.com/viewer?url=${encodeURIComponent(  
    response.data  
  )}&embedded=true`;   
  const openPreviewInNewTab = () => {  
    window.open(viewerUrl, '_blank');  
  };  
  openPreviewInNewTab();  
}
```

Figura 3.11: Deschiderea url-ului în Google Docs

Această funcționalitate are anumite minusuri deoarece nu poate deschide orice fișier, dar este satisfiabilă dacă luăm în considerare faptul că poate deschide cele mai folosite tipuri de fișiere: .pdf, .ppt, .docx, .xlsx.

## Descărcarea

Descărcarea unui fișier se folosește de aceeași metodă ca previzualizarea, dar cu o mică diferență. Link-ul obținut nu mai este deschis în Google Docs, ci este folosit pentru a descărca resursa respectivă.

În schimb, descărcarea unui director implică mai multe etape, fiind o funcționalitate de o dificultate mai mare. În descărcarea unui director trebuie să ne gândim și la alte aspecte precum prevenirea conceptului de „race condition”. Dacă un director se află în procesul de ștergere, atunci utilizatorul nu ar trebui să fie capabil să descarce acel director, inclusiv toate fișierele și directoarele din el. Acest lucru se realizează prin gestionarea a două modele Prisma.

Primul model este CurrentState ce are următoarele atribute:

- id: String = calea resursei.
- userID: String = identificatorul utilizatorului.
- nrRequests: Int = ce reprezintă de câte ori s-a trimis un apel de descărcare asupra resursei.

Al doilea model este CurrentStateOnDelete ce are următoarele atribute:

- id: String = calea resursei.
- userID: String = identificatorul utilizatorului.

Prin urmare, primul pas în descărcarea unui director este de a verifica dacă există fișiere și directoare în tabela CurrentStateOnDelete, adică în procesul de ștergere, ce ar putea interveni în descărcarea corectă a resursei. Dacă există, se afișează un mesaj de eroare utilizatorului, dacă nu există, se trece la următorul pas.

Al doilea pas este de a actualiza tabela CurrentState. Dacă resursele nu există în CurrentState atunci le creăm, dar dacă există incrementăm nrRequests. Astfel obținem un lacăt asupra directorului dorit și a întregului conținut. Motivul pentru care în CurrentState există această variabilă ce necesită incrementare este pentru că un utilizator poate descărca un director și un subdirector al acestuia în același timp.

În continuare, se crează o arhivă, în manieră recursivă, a directorului. Această arhivă va avea un nume de forma: „zip-userIDuser@email.com-numeleDirectorului”. Odată ce arhiva s-a finalizat, actualizăm tabela CurrentState și încărcăm arhiva în

cloud, de unde se va obține link-ul de descărcare. Încărcarea acestei arhive se va face într-un loc pe care utilizatorul nu îl poate accesa din interfața aplicației web.

Pentru a crea arhiva ne folosim de archiver [27] și de modulul fs [28]. Cu ajutorul acestui modul se crează un stream de scriere către arhiva creată și se apelează o funcție recursivă pentru a adăuga directoarele și fișierele din directorul dat de utilizator pentru descărcare. În această funcție se crează directoarele de pe nivelul curent, după care prin folosirea unui stream de citire a fișierelor din cloud, de pe nivelul curent, le adăugăm pe acestea în arhivă. În final se parcurg directoarele create, se construiesc căile respective și se reapelează funcția.

## **Ștergerea**

În ștergerea unui director, principiul de verificare dacă operația poate avea loc este similară cu cea din subcapitolul anterior, diferența constând în schimbarea ordinii. Mai întâi se verifică dacă directorul, inclusiv toate fișierele și directoarele din el, se află în procesul de descărcare sau încărcare, adică fie în tabela CurrentState, fie în tabela CurrentStateOnUpload. Dacă se află, se afișează un mesaj de eroare utilizatorului, în caz contrar, se actualizează CurrentStateOnDelete și începe ștergerea. În final, după ce ștergerea a avut loc, se șterg fișierele și directoarele introduse în CurrentStateOnDelete.

## **Partajarea**

Partajarea de resurse funcționează asemănător cu descărcarea acestora, diferența fiind în gestionarea link-ului resursei.

Când utilizatorul face un apel de primire a unui link către o resursă, acesta se va regăsi în tabelul de link-uri, de unde se poate și copia, oferind de asemenea informații precum data expirării.

Gestionarea acestor link-uri se realizează prin modelul Prisma FileLink, care are următoarele atribute:

- id: String = calea resursei.
- userID: String = identificatorul utilizatorului.
- downloadLink: String = link-ul către resursă.
- validUntil: String = data în care a fost creat.




FILE NAME	EXPIRATION DATE	LINK
Book1.xlsx	6/19/2023, 1:48 AM	
cerere.pdf	6/19/2023, 1:48 AM	
A folder.zip	6/19/2023, 1:48 AM	

Figura 3.12: Tabelul de link-uri

Când un utilizator dorește să vadă tabelul de link-uri se realizează un apel către server pentru primirea acestora. În acest apel se verifică mai întâi dacă există link-uri mai vechi de 24 de ore, iar dacă există le ștergem. După ce are loc această verificare se realizează o prelucrare asupra datii, aceasta fiind reprezentată în baza de date ca un string numeric ce reprezintă timpul Unix. Această prelucrare constă în adăugarea a 24 de ore de la data creării și transformarea acesteia într-un format ce poate fi citit de utilizator.

### 3.2.5 Căutarea

Chiar dacă utilizatorul poate parcurge prin directoare, prin apăsarea acestora, acest lucru se poate dovedi a fi neplăcut dacă acesta are un număr impresionant de directoare, astfel Cloud Storage oferă o funcționalitate de căutare pentru a facilita o experiență mai plăcută.


Search Results 	
FILE NAME	GO TO
Test	→
anothertest.pdf	→

Figura 3.13: Rezultate unei căutări

Odată ce utilizatorul trimite textul introdus în bara de căutare, server-ul caută în

toate directoarele și fișierele acestuia, numele resurselor care includ acel text.

Odată ce server-ul a căutat prin toate resursele utilizatorului, acestuia i se va deschide o tabelă cu rezultatele. Dacă utilizatorul apasă pe butonul din dreapta unui director, acesta va fi dus în interiorul directorului. Dacă apasă pe butonul din dreapta unui fișier, va fi dus în directorul unde se găsește fișierul.

### **3.2.6 Curățarea**

În cadrul acestei teze am vorbit despre o zonă temporară ce este folosită pentru încărcarea, descărcarea și partajarea unui director. Server-ul are o funcție de curățare, care se apelează o dată la două ore și șterge tot ce este mai vechi de 24 de ore din acea zonă. Astfel ne asigurăm că nu avem resurse inutile stocate în cloud.

# Concluzii

## 3.2.7 Direcții viitoare și limitele aplicației

Datorită modului prin care a fost realizată, aplicația web, Cloud Storage, poate fi îmbunătățită relativ ușor și semnificativ.

### Căutare avansată

În prezent, acest proiect nu beneficiază de un instrument de căutare avansat, căutarea fiind bazată doar pe nume, dar acest lucru se poate realiza în urma alocării unor resurse de timp.

### Bară de progres

Când un utilizator încarcă sau descarcă un fișier, ar trebui să existe o interfață ce îi arată progresul, pentru a îmbunătăți experiența acestuia.

### Scalabilitate

Aplicația web, Cloud Storage, nu poate să susțină un număr mare de utilizatori sau de cereri, din cauza modului prin care se face încărcarea. În prezent când o resursă este trimisă către server, aceasta este încărcată în memoria server-ului până când ajunge în cloud. Prin urmare, un număr mare de încărcări poate duce către depășirea memoriei alocate server-ului.

Modul prin care putem combate acest aspect ar fi prin împărțirea fișierelor în bucăți mai mici și trimiterea acestora către server, unde prin utilizarea unui stream le trimitem către cloud. Acest lucru ne-ar permite de asemenea încărcarea de fișiere de o dimensiune mai mare decât se poate în prezent.

De asemenea, dacă am dori să creștem și mai mult numărul de utilizatori ce ar putea să folosească aplicația concomitent, am putea introduce pentru fiecare utilizator

un mecanism bazat pe o coadă de cereri. În prezent, când un utilizator dorește să stocheze multiple fișiere, acestea se încarcă simultan. Acest aspect se poate dovedi plăcut pentru utilizator, dar nu și pentru scalabilitatea server-ului. Prin urmare, dacă am introduce acest mecanism, astfel încât resursele să se încarce pe rând, aplicația ar fi îmbunătățită semnificativ.

### **3.2.8 Opinie personală**

Realizarea unei aplicații web de gestionare a fișierelor în cloud s-a dovedit a fi o muncă grea, ce necesită o serie de cunoștințe pe care nu le dețineam sau stăpâneam în momentul alegerii temei. Fiind primul proiect realizat de o asemenea dimensiune, am întâmpinat diverse dificultăți, primele probleme apărând chiar în stabilirea de tehnologii. Până să ajung la mulțimea curentă, am investit un timp semnificativ în căutarea și alegerea acestora, dorind să am la îndemână unelte cât mai puternice care să facă procesul de dezvoltare mai ușor și rapid.

În final, prin intermediul aplicației web, intitulată Cloud Storage, am reușit să pătrund în tehnicile folosite pentru crearea unei aplicații ce oferă utilizatorului un spațiu online, în care acesta dispune de diverse funcționalități pentru stocarea documentelor sale în cloud. Prin urmare, consider că această teză și-a îndeplinit scopul propus.

# Bibliografie

- [1] Dimensiunea pieței globale. <https://www.fortunebusinessinsights.com/cloud-storage-market-102773>.
- [2] Cota de piață. <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-prov>
- [3] Numărul de utilizatori de aplicații de gestionare a fișierelor. [https://en.wikipedia.org/wiki/Google\\_Drive](https://en.wikipedia.org/wiki/Google_Drive), <https://en.wikipedia.org/wiki/ICloud>, <https://backlinko.com/dropbox-users>.
- [4] Istoria google docs. [https://en.wikipedia.org/wiki/Google\\_Docs](https://en.wikipedia.org/wiki/Google_Docs).
- [5] Typescript. <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>.
- [6] Documentație react. <https://react.dev/learn>.
- [7] React. [https://en.wikipedia.org/wiki/React\\_\(software\)](https://en.wikipedia.org/wiki/React_(software)).
- [8] React dom. <https://www.geeksforgeeks.org/reactjs-virtual-dom/>.
- [9] Chakra ui. <https://chakra-ui.com/docs/components>.
- [10] Axios. <https://axios-http.com/docs/intro>.
- [11] React query. <https://tanstack.com/query/v3/docs/react/overview>.
- [12] Express.js. <https://github.com/expressjs/express>.
- [13] Firebase. <https://firebase.google.com/docs/auth>.
- [14] Prisma. <https://www.prisma.io/docs/concepts/overview/what-is-prisma>.



- [15] Google cloud storage. <https://cloud.google.com/storage/docs/discover-object-storage-console>.
- [16] Google cloud storage, aws, microsoft azure. [https://www.megaport.com/blog/aws-azure-google-cloud-the-big-three-compared/?fbclid=IwAR09sOjM1t4DSzoFjlTv-Q5apPgGVs7xfQQusfQ5ey94MyZrT\\_V9UF0MGdY#features](https://www.megaport.com/blog/aws-azure-google-cloud-the-big-three-compared/?fbclid=IwAR09sOjM1t4DSzoFjlTv-Q5apPgGVs7xfQQusfQ5ey94MyZrT_V9UF0MGdY#features).
- [17] Html drag and drop api. [https://developer.mozilla.org/en-US/docs/Web/API/HTML\\_Drag\\_and\\_Drop\\_API](https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API).
- [18] Datatransfer: items property. <https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer/items>.
- [19] Webkitgetasentry. <https://developer.mozilla.org/en-US/docs/Web/API/DataTransferItem/webkitGetAsEntry>.
- [20] Getasfile. <https://developer.mozilla.org/en-US/docs/Web/API/DataTransferItem/getAsFile>.
- [21] Alegerea directorului din sistemul de operare. <https://developer.mozilla.org/en-US/docs/Web/API/Window/showDirectoryPicker>.
- [22] Obiectul fișier. <https://developer.mozilla.org/en-US/docs/Web/API/FileSystemDirectoryHandle/getFileHandle>.
- [23] Conținutul fișierului. <https://developer.mozilla.org/en-US/docs/Web/API/FileSystemDirectoryEntry/getFile>.
- [24] Obiectul director. <https://developer.mozilla.org/en-US/docs/Web/API/FileSystemDirectoryHandle/getDirectoryHandle>.
- [25] Webkitdirectory. <https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement/webkitdirectory>.
- [26] Signed url. <https://cloud.google.com/storage/docs/access-control/signing-urls-with-helpers#client-libraries>.
- [27] Arhivă. <https://www.npmjs.com/package/archiver>.
- [28] Modulul sistemului de fișiere. [https://www.w3schools.com/nodejs/nodejs\\_filesystem.asp](https://www.w3schools.com/nodejs/nodejs_filesystem.asp).